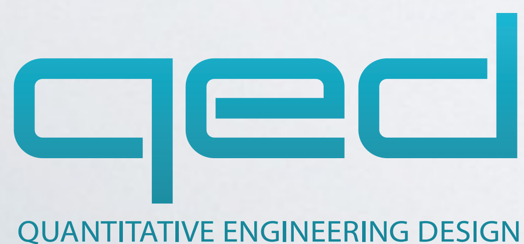
GIT VERSION CONTROL TUTORIAL

William Wu | w@qed.ai
2014 October 7



ABOUT ME

- Scientific Computing Specialist
- background: math, cs, ee
- interests: machine learning, DSP, imaging, data viz, cloud …
- work: various tech companies and research labs
- for more info: http://qed.ai

SOME CORE TOOLS IN MODERN SOFTWARE DEVELOPMENT

- Version Control System (VCS) *

  System that records changes to files over time in order to enable *reversibility*, *concurrency*, and *annotation*.

- Open Source Software (OSS)

- Cloud Computing

- Project Management

OUTLINE

- Why version control? *
- Git basics
- Real-time demo

WHY VERSION CONTROL?

REASON #1: FEARLESS EXPERIMENTATION

REASON #1: FEARLESS EXPERIMENTATION

- Developers maintain many versions of their code to meet rapidly changing demands and save their work.
- Ex: Stable version, experimental beta version, version with different algorithm parameters, version that was done in a rush for the presentation, version from last year, last week, yesterday, …
- Without VCS, you manage this by maintaining many differently named folders, possibly combined with a disk backup system:

proj_v1.1, proj_2007, proj_whyDontThisWork, proj_NewAlgorithm, …

REASON #1: FEARLESS EXPERIMENTATION

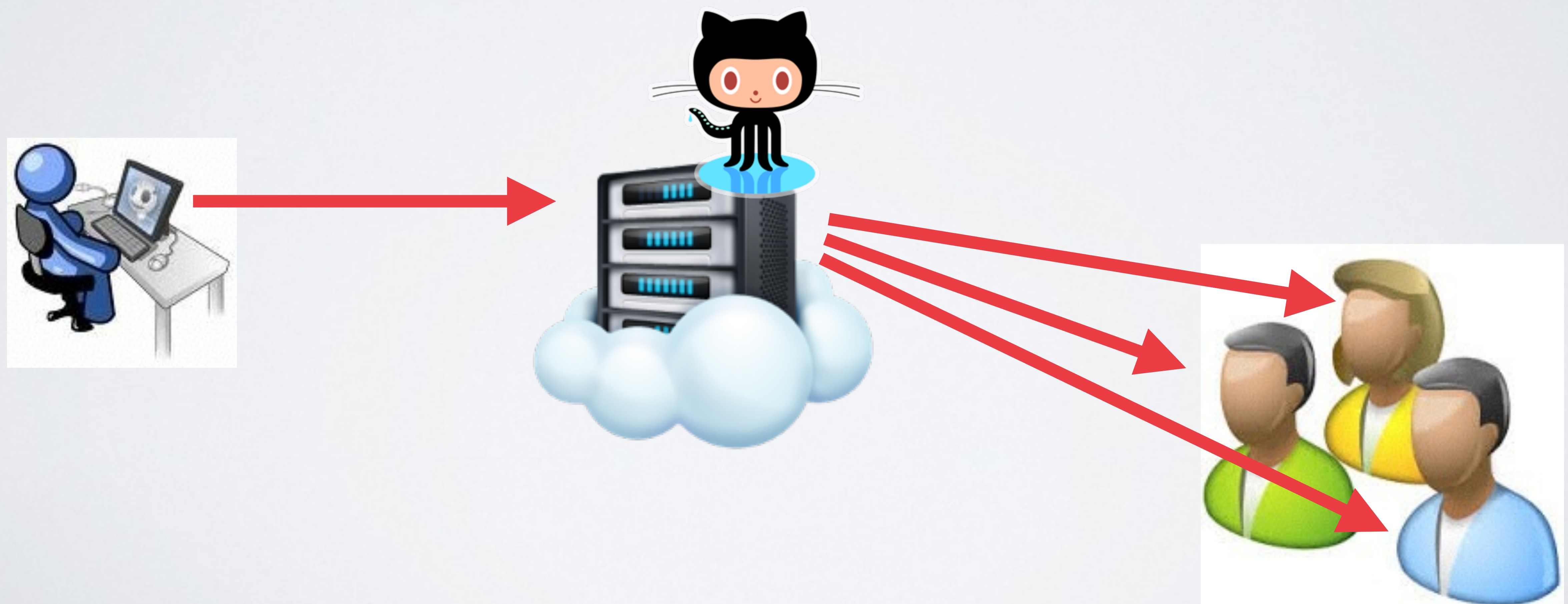
proj_v1.1, proj_2007, proj_whyDontThisWork, proj_NewAlgorithm, proj_Monday5AM, proj_Monday6AM…

- Reality: You can barely maintain this. Consistent naming conventions? Explanations for each folder? Wasting space? Recovering from backup == pain!
- Consequence: **Fear.** You *hesitate* to experiment freely, because every experiment costs management overhead and you are constantly *afraid* of breaking something.

REASON #1: FEARLESS EXPERIMENTATION

- Git dispels your fears.
  - Save whenever you want with ***git commit***.
  - Go back in time with ***git checkout*** *or* ***git reset***.
  - View change history with ***git log***.
  - Switch between different branches with ***git branch***.
  - All magic housed in .git/ subfolder.

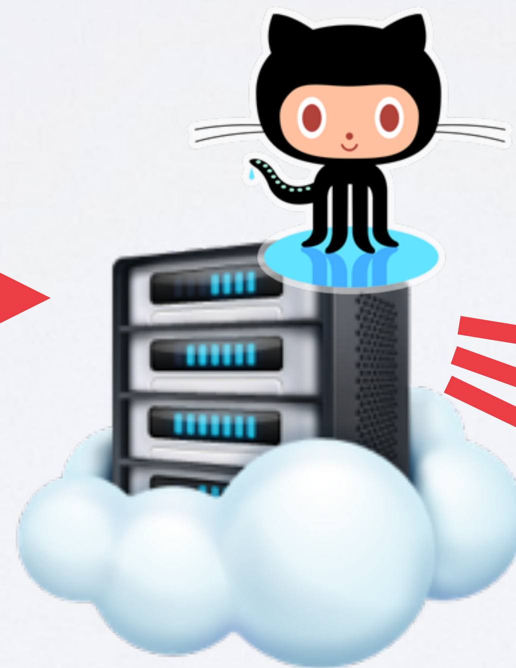
REASON #2: EFFICIENT DISSEMINATION



REASON #2: EFFICIENT DISSEMINATION

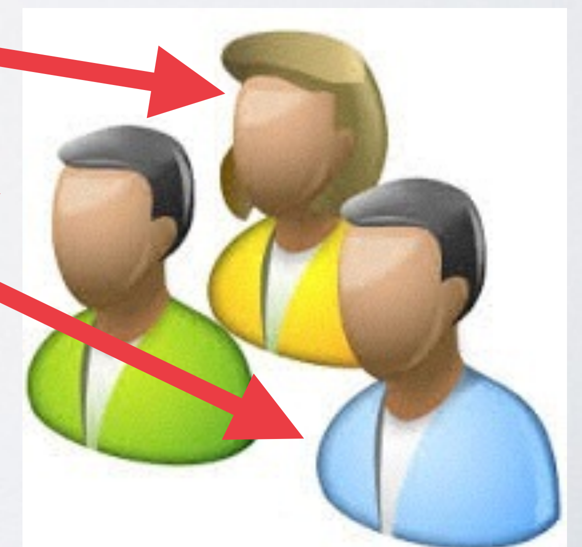
- Software is constantly changing. If it is not changing, it is DEAD.
- Efficient way to distribute updates?

developer:
git push

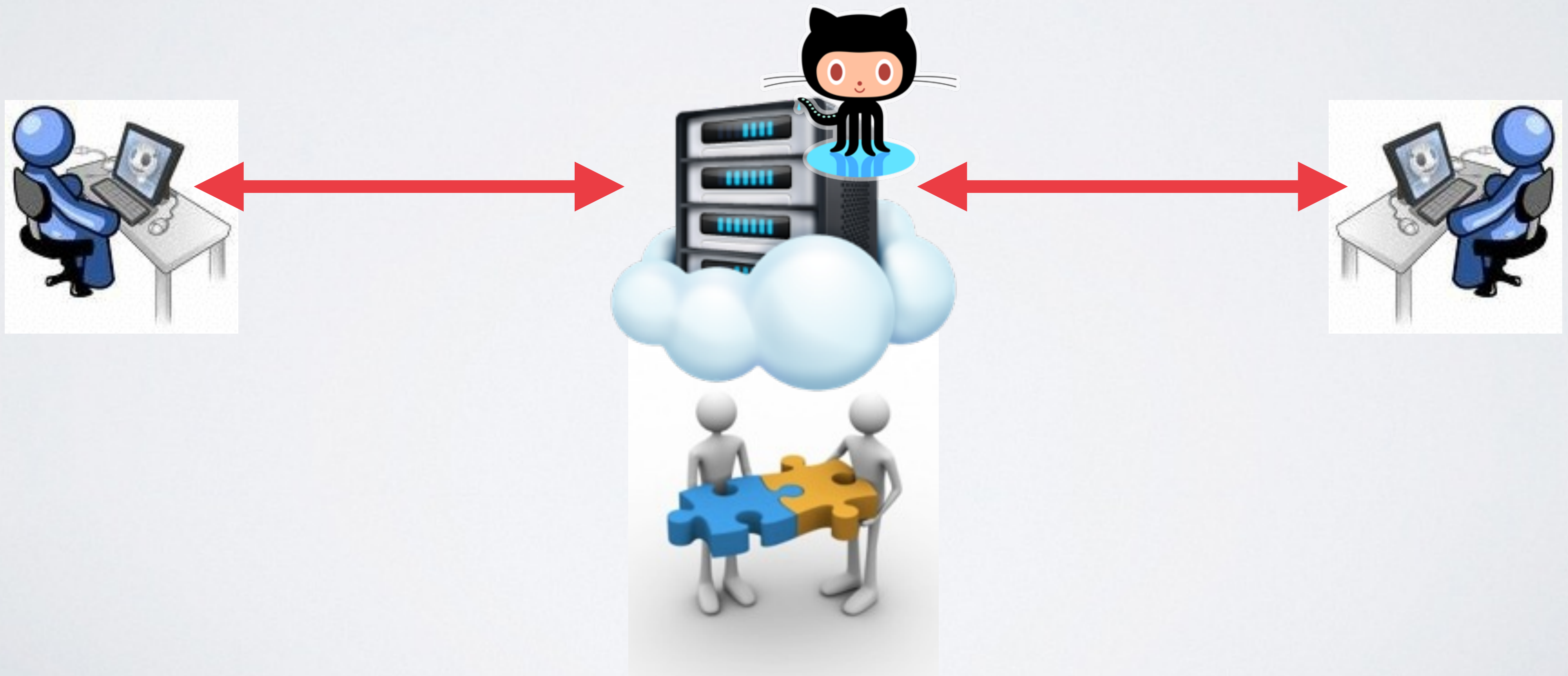


git server

users:
git pull



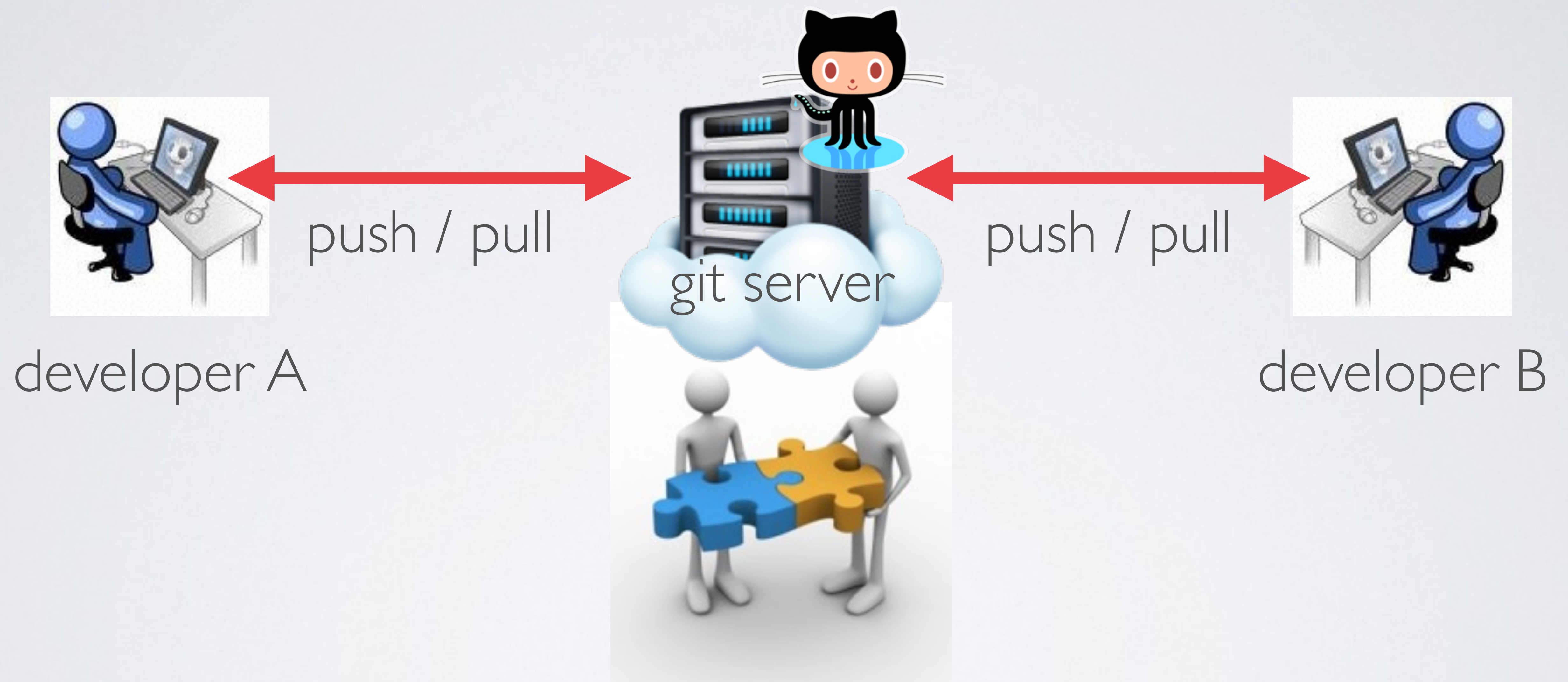
REASON #3:
EFFECTIVE COLLABORATION



REASON #3: EFFECTIVE COLLABORATION

- Version control enables *two or more* people to develop software* together *effectively*.
  - Merging: automated combining of code
  - Tracking: who did what, when, where, and why?
- <u>Those who don't know VCS always work alone.</u>

REASON #3: EFFECTIVE COLLABORATION



- git handles merging code, identifying conflicts, and tracking, so you can work together more efficiently and focus on the interesting parts of coding.

REASON #4: IMPACT

REASON #4: IMPACT

- What exactly is wrong with working alone?

project	started	people	lines	labor years	cost (USD)
R	1993	31	779,588	214	11,769,966
C	1972	314	2,378,985	698	38,384,414
Firefox	2002	3,125	12,454,859	3,888	213,790,957
Python	1991	204	974,479	271	14,855,971
Linux	1991	1,264	341,765	92	5,014,249
Apache	1996	115	1,738,075	487	26,809,851
MySQL	1995	1,253	12,698,247	4,053	222,904,891
PHP	1995	856	2,388,989	686	37,694,087
Android	2008	2,306	10,713,334	3327	183,012,105
WinVista	~2005	~10,000	+50 million	?	10 billion
Ubuntu	1996	543	973,375	264	14,515,251

REASON #4: IMPACT

project	started	contributors	lines	labor years	cost (USD)
R	1993	31	779,588	214	11,769,966
C	1972	314	2,378,985	698	38,384,414
Firefox	2002	3,125	12,454,859	3,888	213,790,957
Python	1991	204	974,479	271	14,855,971
Linux	1991	1,264	341,765	92	5,014,249
Apache	1996	115	1,738,075	487	26,809,851
MySQL	1995	1,253	12,698,247	4,053	222,904,891
PHP	1995	856	2,388,989	686	37,694,087
Android	2008	2,306	10,713,334	3327	183,012,105
Ubuntu	1996	543	973,375	264	14,515,251

- **All made possible through version control and OSS.**
- Effective crowdsourcing yields *speed*, *strength*, *security*, *outreach*, *feedback*, and *sustainability*.
- Ubuntu "the belief in a universal bond of sharing that connects all humanity." (Nguni Bantu)
- git connects you to OSS and unlocks possibility for **real impact**.

GIT BASICS

HISTORY

- Many VCS options since 1972: SCCM, CVS, Subversion, BitKeeper, Mercurial, git, etc …
- Git is currently the most popular modern VCS.
- Created by Linus Torvalds (Linux); git = British English slang for 'unpleasant person'
- Design features:
  - More efficient and scalable than other VCS
  - More flexible and fast branching and merging
  - Snapshots directory trees of files
  - Cryptographic authentication history
  - Can run in decentralized mode (no shared server)

BASIC WORKFLOW

- Developers JC and Willy are doing a project together. JC first ***initiates*** a ***git repository*** that is hosted on a ***shared server***, such as github. Willy then ***clones*** the repository.
- JC and Willy assign their tasks for the week, then work independently.
- JC finishes first. She ***adds*** any new code files, ***commits*** her code (save point), ***pushes*** her commit to the server ("origin" by default), and goes to bed.

[JC's computer]

$ **git init**

$ **git remote add origin git@github.com:group/proj.git**

$ **git push origin master**

[Willy's computer]

$ **git clone** git@github.com:group/proj.git

[jira, asana, trello, basecamp, etc.]

> "backend" assigned to JC

> "frontend" assigned to Willy

[JC's computer]

$ **git add** *.R *.php

$ **git commit** -a -m "finished backend"

$ **git push origin master**

BASIC WORKFLOW

- Willy finishes his task four days later. He doesn't remember everything he did since then, so he uses a ***diff*** to review his changes first. Then he ***adds*** any new files, and makes a ***commit*** with a comment describing what he did.
- Willy learns that his local code is outdated, so he ***pulls*** the latest updates. Git locally merges JC's changes with Willy's changes, on Willy's machine. Then Willy ***pushes*** newly merged code to the server.
- Next day, JC pulls latest code from the server. Everyone is in sync!

[Willy's computer]

$ **git diff**

$ **git add** *.html *.js *.css tiny-images/*

$ **git commit -a -m** "finished frontend; using jQuery and Twitter Bootstrap"

[Willy's computer]

$ **git pull origin master**

$ **git push origin master**

[JC's computer]

$ **git pull origin master**

CORE OPERATIONS

initiate repo
git init

clone repo
git clone [git-url]

add files to version control
git add [files]

make a save point
git commit -a -m "short message"

push your committed changes to the server
git push origin master

pull the latest changes from the server
git pull origin master

review unsaved changes (useful before commit)
git diff and **git status**

review commit history and comments
git log

determine who to blame for each line of code
git blame [filename]

move a file under version control
git mv [oldname] [newname]

remove a file
git rm [filename]
git rm —cached [filename]

BRANCHING OPERATIONS

create new branch based on current branch

git checkout -b [branch-name]

view available branches

git branch

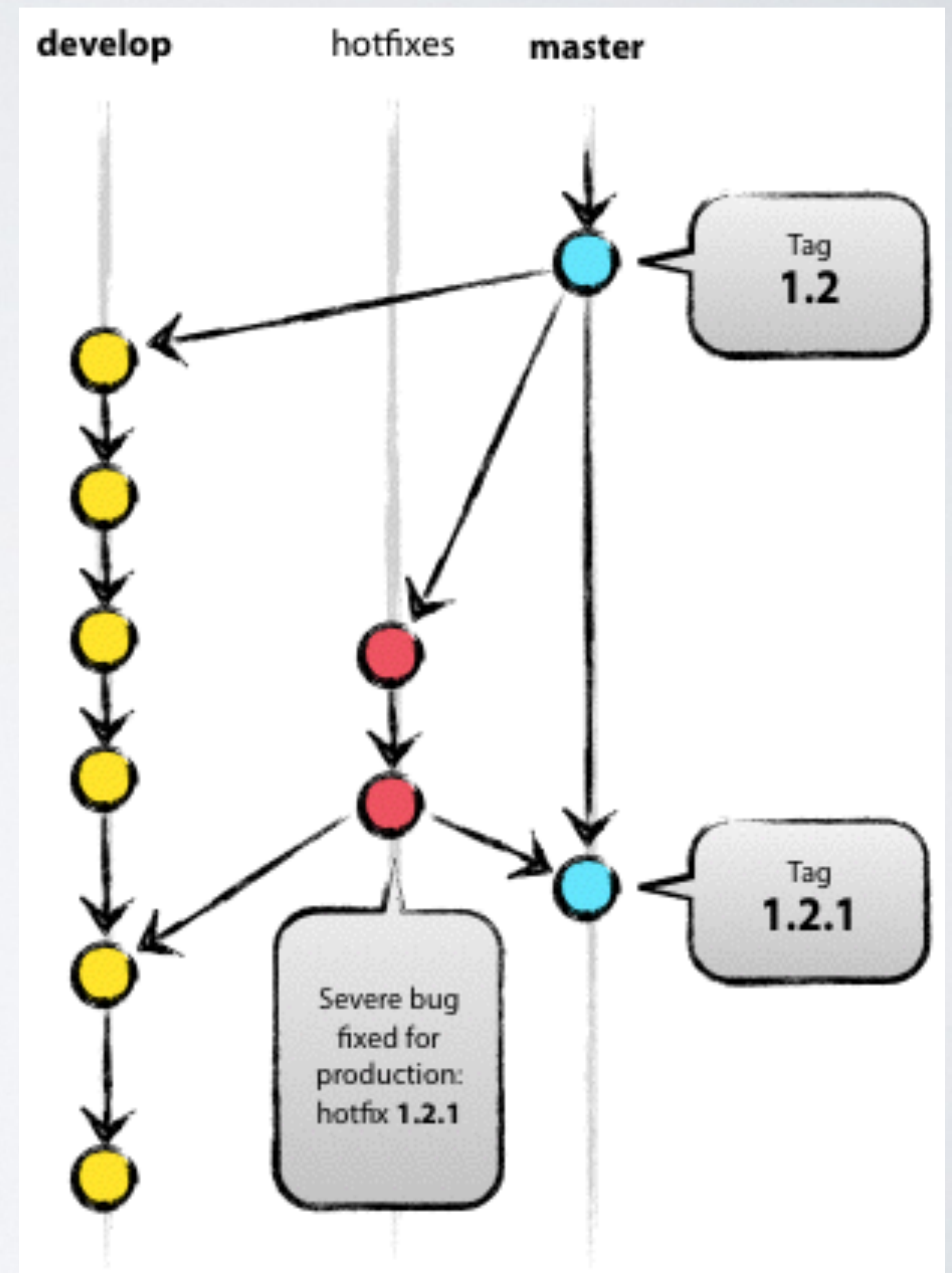
switch branch

git checkout [branch-name]

merge changes from one branch into another; e.g. merge new commits on branch "dev" into branch "master":

git checkout master

git merge dev



LESS COMMON OPERATIONS

checkout previous time point

git checkout [SHA1-hash]

(where hash is retrieved from git log)

reset branch to particular time (irreversible)

git reset —hard [SHA1-hash]

save uncommitted changes

git stash

restore uncommitted changes

git stash pop

IMPORTANT RULE:

NEVER ADD LARGE BINARY FILES TO GIT!

check out branches available on server that are not yet being followed locally:

git fetch -a

git branch -a

git checkout [remote-branch]

assign a tag to current version

git tag -a [tag-name] -m [message]

set attributions

git config user.name [your-name]

git config user.email [your-email]

SETUP

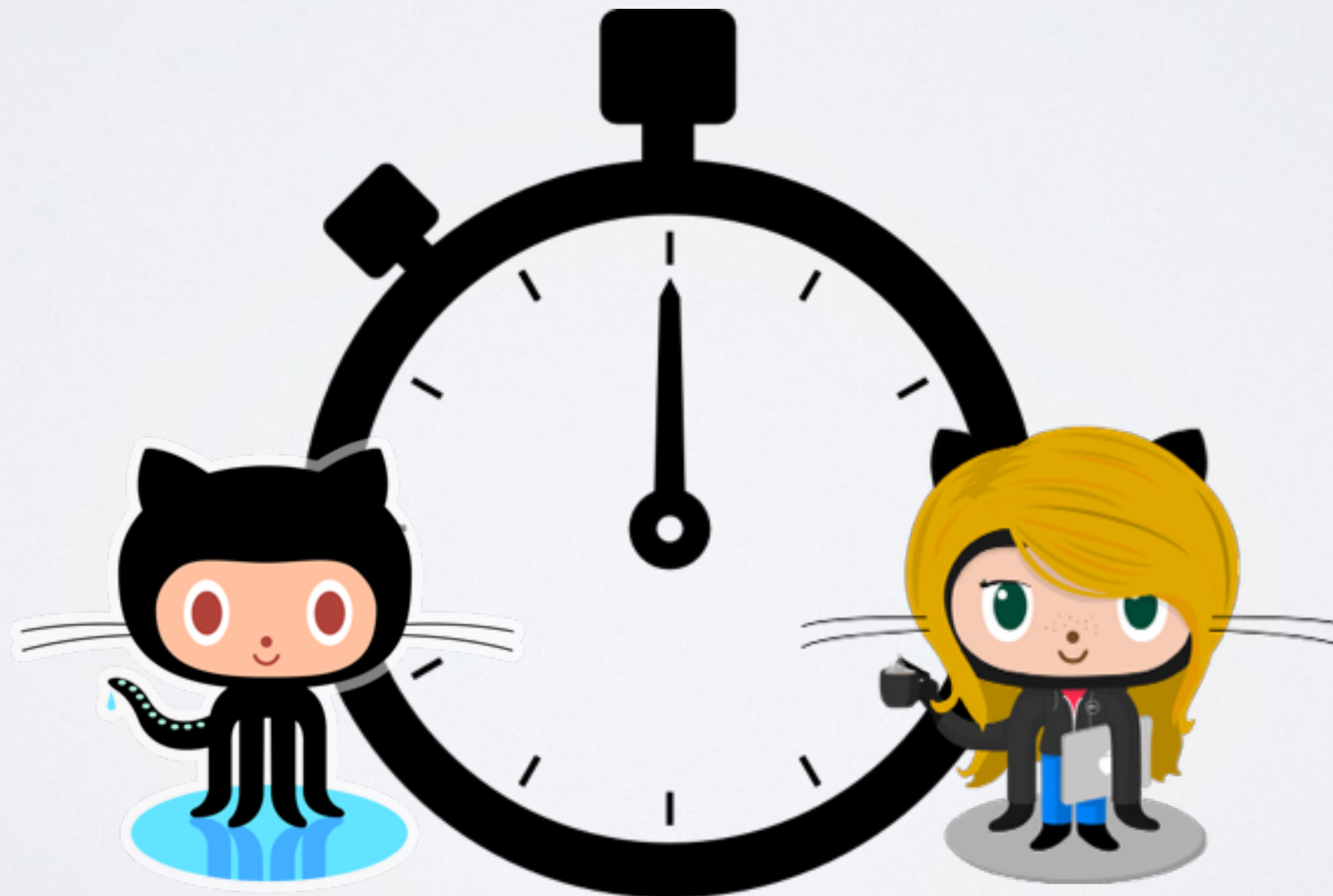
- Server-side:
  - Web services: **github.com**, bitbucket.org, code.google.com, sourceforge.net, …
  - Host your own: **gitolite**, gitlab, gerrit …
    - Advantage: infinite # private repos at no extra cost
    - Disadvantage: adminstration of your own server
  - If you don't mind making code public, or require only a few private repos, then using github.com is a simple solution.

SETUP

- Client-side:
  - Windows: http://msysgit.github.io/
  - Mac OS X: brew install git
  - Linux: sudo apt-get install git
- SSH Keys:
  - https://help.github.com/articles/generating-ssh-keys/
  - Windows: http://eureka.ykyuen.info/2010/05/19/git-generate-public-key-for-github-using-msysgit/

REAL-TIME DEMO

Now we'll pose a problem and demonstrate collaboratively programming a solution in real-time. Multiple devs will discuss the problem, assign subtasks, and then use git to share the work.



REFERENCES

- Git Book: http://git-scm.com/book/en
- Interactive tutorial: https://try.github.io/
- Graphical tutorial: http://www.wei-wang.com/ExplainGitWithD3/
- Cheatsheets: https://www.google.com/search?q=git+cheatsheet
- More details: http://gitready.com

THE END

- Try using git in your next project!
- Once you start using it, you'll wonder how you ever lived without it!